Dr. Douglas W. Gage
Information Processing Technology Office (IPTO)
Software Programs for Robotic Autonomy

The robots we see in films and other media are powerful machines that effectively and confidently pursue their goals, whether for good or evil, while they move comfortably through the real world. Current real-world robots fall far short of this image. But two DARPA software programs are working to bring useful robots closer to reality. But these are technology efforts, not system-level development programs. Our focus is not on building robots, but on making them smarter.

Software for Distributed Robotics (SDR) is developing robot behavior and coordination software to allow large groups of extremely resource-limited small robots to perform useful tasks in the real world. We are also developing tools to permit human users to task and query a system of robots as a single unit, without considering what any individual robot is doing.

Mobile Autonomous Robot Software (MARS) is an effort to exploit machine learning and perception in order to achieve perception-based autonomous vehicle navigation, with performance at near-human levels over a wide range of real-world environments and to provide effective interaction with humans in various roles. The MARS approach is to exploit operator intervention and other tools as "crutches" to achieve revolutionary autonomous capabilities.

The focus of both programs is to develop robotic software technologies that provide functional capabilities beyond the requirements of any specific application or mission. These generic capabilities often correspond to the basic perception and common sense reasoning abilities that every human, or animal, is born with or develops. Because each of us does these things so effortlessly, the need to implement them explicitly in our robots is often overlooked when analyzing the requirements of any specific application. A key element of these programs is to articulate those explicit functional and performance requirements that often remain implicit and unrecognized.

Many in the robotics community believe that some tasks can be performed most effectively by a large number of small, inexpensive, mobile robots. The concept is inspired in part by the perceived industry of ant or bee colonies. A body of research exploring the issues in creating and exploiting swarm systems has been developed over the past 10 to 15 years. To date, however, no distributed robotic systems have been deployed, and physical systems of mobile robots, even in a research environment, have been limited to no more than 20 or 30 units.

It is possible to have systems with a small number of small robots, or a large number of large robots, and SDR technologies can be used in these kinds of systems. The ultimate SDR vision, however, is to realize systems with very large numbers of very small robots. Employing large numbers of robots can provide system-level functionality and performance in several ways:

- A search for stationary targets can generally be done faster using multiple searchers.

- A search for targets capable of movement might require multiple robots in different places to prevent the targets from evading detection indefinitely.

Other examples are building a network of communications relays, providing continuous detection of intruders at multiple but physically separated entry points, or simply lifting the four corners of a large object simultaneously to move it.

- Using multiple devices reduces the risk of failure in performing tasks that could result in a robot being disabled or destroyed. An example is the "Pick Up and Carry Away" approach to cleaning up unexploded cluster munitions.

- Some missions might require that one device be dedicated to each of a potentially large number of specific places. An example is the "Blow in Place" method for cleaning up unexploded ordnance by simultaneously detonating an explosive charge placed next to each weapon.

To realize these benefits provided by a system of many robots, SDR is developing technologies for group behaviors, coordination, communications, and user interfaces for tasking, monitoring, intervention and assessment. These are the challenges of *many*, which provide the development agenda for SDR.

Making our robots small can also provide important payoffs. Small robots are lightweight and easy to transport—especially important if the system includes many robots—and they can fit into small spaces inaccessible to larger devices. The challenge is to assemble a system capable of performing useful missions by using many small robots, each limited in strength, power, communications, sensing, and processing abilities. These challenges of *small* provide constraints on the SDR development agenda.

The behavior of the system as a whole depends on the behaviors of each individual robot, This relationship can be viewed in several ways.

- In the first view, desired system behavior can be partitioned into subbehaviors, or tasks, and these subtasks allocated to the various robots. An SDR effort at Carnegie Mellon University explores an innovative allocation method using an economic model in which robots bid for task assignments. The winner has the option of further partitioning the task and re-auctioning the subtasks.

- In the second swarm view, the desired system-level behavior is seen as "emerging" when individual robots are brought together, and there is no explicit task allocation. A number of SDR efforts, including one at Sandia National Laboratory, explore this paradigm, using analogies with biological or physical processes such as gaseous diffusion. These analogies with natural processes are critical tools for the distributed system developer, since we do not know in general how to design individual behaviors that will achieve a specified system-level goal.

In any event, each robot's actions can be based only on the information available to it through its sensors, communications channels, and memory storage, which might be extremely limited. For this reason, a final critical input to behavior is randomization—the ability to choose between behavior options based on throwing dice. This allows the ensemble to harness the statistical power of large numbers to achieve its goals, but it also means that a specific aspect of system performance (for example, finding every single mine within a specified area) cannot be guaranteed. The challenge is to ensure that the system developer and system user understand the statistics.

Also, since SDR is focused on the development of software, it is important that we explicitly identify to system-level developers exactly what hardware resources we require for the behaviors we develop so the necessary sensors, communications, memory, and processing will be available on deployable hardware.

SDR efforts address both explicit and implicit communications.

- We are developing networking protocols that minimize the energy needed to provide traditional wireless networking connectivity. That's the ability of one node to send a message to another node while being smart about when to transmit a message, how much power to use for each transmission, when to turn on a receiver, and how many hops to use to route each message. A topic for future exploration is the exploitation of the synergy between the information the network node software must maintain to support connectivity and the information the robotic controller maintains about its physical environment.

- We are also investigating nontraditional modes of communications, including the use of "pheromone" packets to provide an indication of physical network topology that can be used as the basis for behaviors.

A distributed robotic system must provide a coherent model of its functionality and performance. This permits a user to determine what it can do, when to use it, how to configure it, how to monitor and control its activity, and how to determine what it has accomplished. While this can be done with a traditional command-based user interface, SDR is also exploring implicit user interfaces. For example, HRL Laboratories is developing a scheme where information from each robot can be displayed to a soldier on a wearable display. If a swarm of robots has flooded a building and located a target of interest, the soldier would see an arrow displayed over each robot pointing in the direction of the target.

Besides the obvious technical challenges, the process of developing a distributed robotic system contains some subtle "gotchas."

- If the system's operation depends on swarm behavior, one possible problem is that meaningful ensemble behaviors might fail to emerge.

- A second possibility is that the behaviors that do emerge might be wrong and not produce the required system-level functionality.

- While simulation is necessary to support the initial design of behaviors, testing realistic numbers of physical robots in real-world environments is absolutely required to determine whether the interaction of software with physical sensors and communications is the same in the real world as it is in simulation.

- Any deployed system will likely employ tightly integrated, minimum cost, mass-produced robots. These certainly will be expendable and probably deliberately disposable after a single use. This implies nonrechargeable primary batteries and communications resources that provide only those capabilities required by the application itself, which may be minimal to nonexistent.

- The "gotcha" is that the testing required to develop a deployable system will require robots that can be reused; this means rechargeable batteries, communications and developer interface tools to support problem diagnosis, plus the ability to reprogram the robots in the field. The greatest challenge, therefore, is not the system as it is to be deployed; rather, it is the system as it must be configured to support the development process.

Over the next 2 years, the SDR Program will experiment with testbed distributed robotic systems to assess mission-level effectiveness in performing indoor reconnaissance, an application of increasing importance for military operations in urban terrain, special operations, and counterterrorism. Multiple mission scenario experiments will address multiple types of targets and tasks. Examples of targets are humans, sound sources, and chemicals. Tasks might include covertly locating a single target, locating places where targets are concentrated, or ensuring that there are no targets in a particular area. Lessons learned from these experiments should identify the most promising approaches for achieving distributed robotic capabilities in deployed systems. The design space for distributed robotic systems is extremely large, and the best solution depends sensitively on the application. Thus, it is very important to be able to specify the precise details of the desired capability.

How many robots does it take to be "many”? And how small must a robot be to be "small”? Different communities use the same word with totally different meanings. For example, the Tactical Mobile Robotics Program addresses the development of small robots, by which they mean 20 pounds or less. The Future Combat Systems Program addresses the development of small unmanned ground vehicles, by which they mean "able to be transported in a C-130," which is 20 tons or less. The SDR Program envisions hundreds or thousands of robots, each less than 1 pound and potentially much less.

The MARS (Mobile Autonomous Robot Software) Program addresses the development of several technology areas to achieve autonomous capabilities for individual robots. These include machine learning, machine perception, and the interaction of robots with humans. While MARS was initiated to develop generic robotic software technologies, the prominence of the Future Combat Systems Program as an obvious transition target has made the autonomous navigation of unmanned ground vehicles a primary

focus for the MARS Program. MARS also addresses other applications including small mobile robots capable of indoor operations, humanoid robots, and heterogeneous teams of small ground and air platforms.

Autonomous navigation is important for unmanned vehicles because the alternative—teleoperation by a remote operator—is expensive in terms of both operator attention and communications bandwidth and simply doesn't work well in many situations. A remote driver generally can't drive as fast as a vehicle's performance allows and, even at reduced speed, risks bumping into obstacles, falling into a ditch, or, in the case of small robots, getting completely lost.

Serious DoD efforts to develop unmanned ground vehicle autonomy began in the early to mid-1980s with DARPA's Autonomous Land Vehicle Program and have continued. While great progress has been made, unmanned vehicles are still not fully capable of driving themselves. The challenges that remain are difficult. The key problem is perception (machine vision), which appears to be one of those *really* hard problems like understanding speech and translation of natural language.

MARS will not solve autonomous driving in the next 2, 4 or even 8 years. Instead, we are focusing on developing tools that can be incorporated into a system to achieve two goals: to support the rapid deployment of semi-autonomous capabilities, and to accelerate the evolution of fully autonomous operation over the next decade or two. Here are some examples of the sorts of tools that will help us learn what it is we need to know to achieve these goals:

- When an autonomous unmanned ground vehicle encounters a situation it cannot handle, it must rely on operator intervention. This often means some sort of emergency stop, followed by teleoperation. The result is that the robot's sensors and perception software don't get to experience the greatest environmental challenges. The system software must be structured so the robot continues to experience its operating environment even when the human operator intervenes.

  Operator intervention may be required at a number of different levels, including high-level planning, behavior selection, and perception. In each case, the software must provide the operator with the information needed to intervene effectively and efficiently. In some cases, the robot may need to maintain an explicit model of the operator's task loading and performance. In all cases, the interface design must reflect the operator's needs and capabilities.

  Operator intervention should be exploited by the system to support the evolution of autonomy in at least two ways: by providing input to machine learning to support system adaptation, and by characterizing those situations when operator intervention is necessary because autonomous capabilities fail. This will help establish priorities both for near-term system tailoring and for long-term research.

  Finally, measuring the amount of required operator intervention provides a metric for judging the system's level of autonomy—the less intervention, the higher the level of autonomy. The MARS Program goal is to double the average time between required operator interventions and cut in half the average operator time required for each intervention. MARS efforts have advanced development of a number of machine learning techniques (supervised, reinforcement, and imitative) and applied them to robotic autonomy in behavior selection, behavior parameter tuning, and perceptual classification. Current MARS efforts in machine learning focus on structured experimentation to assess and validate specific techniques in specific system roles.

- Tools that support the systematic assessment of perception and behavior performance in terms of quantitative metrics are absolutely critical to the success of an evolution-based approach to the development of autonomy. The degree of required operator intervention constitutes one measure of system autonomy, but other metrics are needed at the subsystem level to identify specific research objectives that carry the highest performance payoff.

- Every movement of every vehicle represents an opportunity to gather information about the path it follows. If this is done effectively, the information can be used by other autonomous vehicles to

travel along that same path. The key is gathering, processing, and storing the information in a way that permits it to be used later. A perception-based representation of the path is required, recording the vehicle's actions, the characteristics of the environment, or both, at the highest possible level of abstraction.

- Robots must be able to interact with humans in different modes. For example, a distant commander might need a high-level command interface and a "God's-eye" view of a group of autonomous vehicles, while a remote operator requires a much more detailed interface to a single robot, including high bandwidth teleoperation. MARS addresses contact with humans in the robot's environment, including teammates, bystanders, and adversaries. For example, Robonaut must interact with human astronauts, while a vehicle driver must interact with other drivers and pedestrians. Achieving human-equivalent interaction capabilities is a potentially critical technology challenge for the deployment of autonomous robots.

- Development of sensory perception for unmanned ground vehicles traditionally has focused on obstacle detection, terrain classification, and the assessment of terrain traversability. Meanwhile, sensory perception for weapons systems has focused on automatic target recognition. MARS focuses instead on sensing, interpreting and understanding environmental features, including humans. It uses model-based approaches and fuses the data from multiple sensors to detect, classify, identify, and track both static and moving objects.

The second half of the MARS Program focuses on the integration of these tools to support system-level experimentation. The idea is to validate the MARS evolutionary approach, assess the contributions of each tool to overall capabilities, and identify the technology areas offering the greatest promise for future development.

In addition to many technical challenges, developing software for robotic autonomy also presents important management challenges.

An autonomous robot—especially an unmanned ground vehicle system—is a complex beast, and this is reflected in the system development process. Achieving the desired mobility goals focuses attention on the vehicle chassis, engine, transmission, and suspension. Then sensor, computer, and communications hardware must be procured or implemented and integrated into the system. Program management focuses on hardware tradeoffs, procurement actions, and delivery dates. And, as the once-distant date of the promised system demonstration looms ever closer, test plans must be developed and test facilities prepared.

Thus, it's easy for software, especially autonomy software, to get lost in day-to-day program management activities. And, of course, software is usually viewed from a software engineering perspective. This means selecting and implementing the software development infrastructure, designing program modules, writing lines of code, unit test, and software integration to make sure that all the right messages flow between the right modules at the right times. The problem is that neither the hardware-oriented systems management view nor the conventional software engineering view explicitly addresses the interaction of a robotic vehicle with its environment, the cognitive domain. This depends on the subtle interplay between the functionality and performance of multiple hardware sensors and perception and behavior software modules. Successful autonomous behavior means the robot shouldn't bump into the tree in front of it, not just that all the messages flow correctly between the software modules. The tools we are developing in MARS are essentially designed to make the imperative "don't hit the tree" an explicit goal for the entire system design process.

Thank you.